Phase 3 Report

Interactions between different components of our system

Feature/Interaction	Test Case/Class	File
Set and Get Score of main character	testScoreGetterAndSetter()	MainCharacterTest.java
Get the sprite of the main character	testGetPlayerSprite()	MainCharacterTest.java
Update main character based on direction	testUpdateMoveUp/Down/left/Right()	MainCharacterTest.java
Set and Get the damage amount of a moving enemy	testDamageAmountGetterAndSetter()	MovingEnemyTest.java
Spawn a moving enemy	testSpawning()	MovingEnemyTest.java
Move the moving enemy toward the main character	testMoveTowards()	MovingEnemyTest.java
Update the moving enemy	testUpdate()	MovingEnemyTest.java
Set reward amount of a bonus reward	testSetRewardAmount()	BonusRewardsTest.java
Spawn a bonus reward	testSpawning()	BonusRewardsTest.java

Get the sprite of the bonus reward	testGetBonusRewardsSprite()	BonusRewardsTest.java
Check the collision of the bonus reward	testCheckCollision()	BonusRewardsTest.java
Claim the bonus reward	testClaimReward()	BonusRewardsTest.java
Get and set the static enemy’s visibility	testVisibleGetterAndSetter()	StaticEnemyTest.java
Get and set the static enemy’s damage amount	testDamageAmountGetterAndSetter()	StaticEnemyTest.java
Get and set the static enemy’s detection status	testIsDetectedGetterAndSetter()	StaticEnemyTest.java

Spawn the static enemy	testSpawningInBounds()
	
	testSpawningOutOfBounds()
		StaticEnemyTest.java

Get sprite of the static enemy	testGetStaticEnemySprite()	StaticEnemyTest.java
Check collisions of the static enemy	testCheckCollision()	StaticEnemyTest.java
Update the static enemy	testUpdate()	StaticEnemyTest.java

Ensure that the main character’s score is reduced	testPunishmentNegativeTrue()
	
	testPunishmentPositiveFalse()
		StaticEnemyTest.java

Set reward amount for a static reward	testSetRewardAmount()	StaticRewardTest.java
Spawn a static reward	testSpawning()	StaticRewardTest.java
Get the sprite of a static reward	testGetStaticRewardSprite()	StaticRewardTest.java
Check the collision for a static reward	testCheckCollision()	StaticRewardTest.java
Claim a static reward	testClaimReward()	StaticRewardTest.java
Draw a static reward	testDraw()	StaticRewardTest.java
Set up the game screen size and background color	testScreenSetUp()	GameEngineTest.java
Painting of the game per frame	testPaintComponent()	GameEngineTest.java
Reset the game	testResetGame()	GameEngineTest.java
Update the status of both the moving and static enemy	testUpdateEnemies()	GameEngineTest.java
Draw the map of the game world	testDrawMap()	GameWorldTest.java

Check the collision status of a cell	testCheckCellsNoCollision()
	
	testCheckCellsCollision()
		CollisionDetectorTest.java

Make sure the main character don’t go out of bounds (off the map)	testCheckCellsOutOfBounds()	CollisionDetectorTest.java

Behavior of a portal cell in the game world	testCheckCellsPortal()	CollisionDetectorTest.java
When main char has 500 points, the portal will open	`testCheckCellsPortalOver500()`	CollisionDetectorTest.java

Line/Branch Coverage of Tests

- The line and branch coverage for the GameWorldTest were both 100%.
- The line and branch coverage for the staticEnemyTest are both 100%
- The line coverage for the GameEngineTest was ~90%, while branch coverage was ~ 80%. Some private methods, as well as exception lines, we were not able to fully cover.
- The line and branch coverage for the StaticRewardsTest were both 100%.
- The line and branch coverage for the BonusRewardsTest were both 100%.
- The line and branch coverage for the collisionDetectorTest were both ~85%. Since our maps have different collisionDetector(obstacles and collision walls) the coverage changes according to which maps we are testing.

Takeaways From Testing

Although few changes were made to the code for the game during the testing phase, we saw that our tests had limitations given that the methods for the different functionalities of our game were often quite complex. This resulted in a less than 100% line coverage as some of the lines were difficult to reach in a test method, while branches were mostly covered as not many conditions were complex in a testing environment. Quality of our code was improved as many of our methods were refactored. Testing was conducted after this and did not result in many defects being found in our app.

To improve testability, we needed to convert some private tests to public.